

Streamdice: An encryption algorithm based on catalogued shuffled keyboards

Andrew R. Garcia
garcia.gtr@gmail.com

Abstract

ImageMesh is a method available in versions ≥ 2.0 of the voxelmap Python library that generates 3D models from images using Convex Hull in 3-D to enclose external points obtained from a series of partitioned point clouds. These point clouds are generated by assigning the relative pixel intensities from the partitioned images as the depth dimension to the points. In this paper, we describe the limitations of the original ImageMesh method and the quick solution we have implemented to address them. Additionally, we introduce MeshView, a Python visualization tool developed in tandem with ImageMesh that provides a convenient way to visualize the 3D models generated by ImageMesh. Finally, we discuss the GPU memory space complexity of both methods.

1 Introduction

Good encryption can be used to protect data and private information. When properly encrypted, even if data is accessed in an unauthorized manner or unwillingly disclosed, the non-consented reader will be unable to read it without the correct encryption keys. The algorithm presented here, **streamdice**, is a stream cipher which encrypts characters (i.e. letters, numbers and some allowed signs) by both their specific identity as well as their relative location in the message thread. For streamdice, the stream units are shuffled keyboards generated by a pseudo-random number generator (PRNG), each of which are shifted once for every single encrypted character. The shuffled keyboards are limited and kept in memory with hashes, which are in turn dependent on the provided keys for encryption. The pseudo-random factor obfuscates the periodicity of the algorithm, and the encryption operations make it challenging to exploit by brute force.

2 Method

This method uses a hashmap where each one of its keys corresponds to an index of the keyboard character representing it in QWERTY order. For instance, the

first keys for Q, W, and E characters are 1, 2 and 3, respectively. This ordered arrangement is analogous to a symbolic keyboard as the one in Figure 1.

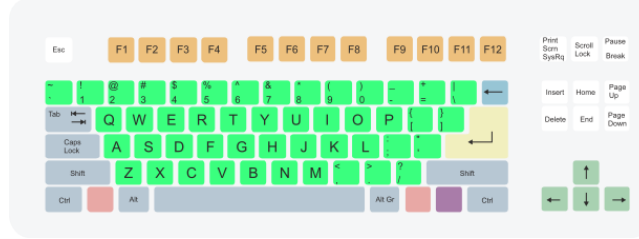


Figure 1: Standard QWERTY keyboard

The character values of the hashmap are shuffled with a pseudo-random number generator (PRNG) seeded by a "hash" number. For instance, shuffling the original keyboard in Figure 1 with a Mulberry32 PRNG and a #5443 hash will give the shuffled keyboard in Figure 2. Because the values are shuffled, the keys will be preserved, and thus the ciphertext can be transformed back by reference to the original keyboard, i.e.

$$s^{\wedge}f)/d)zxb \rightarrow [4, 18, 1, 7, 15, 14, 6, 15, 18, 3, 5](indices) \rightarrow dragonforce$$

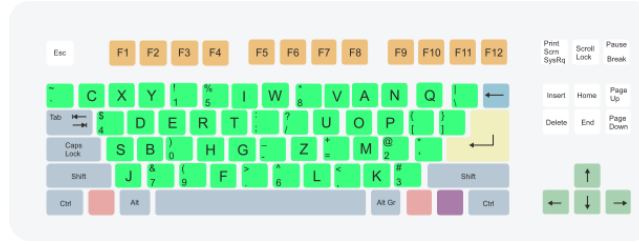


Figure 2: Randomly-shuffled keyboard with #5443

For the streamdice algorithm, however, the keyboard is shuffled each time a new character is encrypted with N permutations, calling a new shuffling operation by its PRNG hash as shown in Figure 3. As seen in the same figure, the permuted keyboards repeat periodically if the number of keyboards is less than the number of characters to encrypt. The specific hashes are computed directly from the 2 keys provided by the user for the encryption. An \mathbf{H} vector contains all the H_i hashes used to generate the shuffled keyboards. The N number of H_i hashes is equal to the number of digits provided for key_2 and are computed in the following way:

$$\theta_i = (key_2 // 10^i) \% 10 \quad \text{and} \quad H_i = key_1 + \theta_i$$

The hashes used to generate the new keyboard, rather than the specific keyboard arrangement, are the objects kept in memory throughout the encryption.

As suggested above, the decryption takes the keys used to encrypt the messages and reverses the protocol. This method, thus, optimizes auxiliary space, $\mathcal{O}(N)$, rather than encryption time complexity, $\mathcal{O}(MC)$, where N is the number of digits of the key_2 , while M and C are the message length and number of keyboard characters to encrypt, respectively.

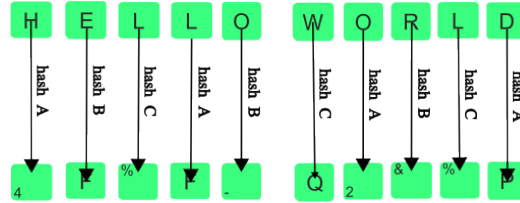


Figure 3: Encrypting *hello world* with a periodically-repeating stream of 3 shuffled keyboards.